Discussion 5: Mutability, OOP

disc05.pdf (disc05.pdf)

This is an online worksheet that you can work on during discussions. Your work is not graded and you do not need to submit anything.

Walkthrough Videos

Feel free to try these problems on the worksheet in discussion or on your own, and then come back to reference these walkthrough videos as you study.

> To see these videos, you should be logged into your berkeley.edu email.
>
> **Note:** These videos are from a past semester and are not updated to reflect any changes (variable names may be different). Also note that we unfortunately do not have a walkthrough video available for "Add This Many."

61A SP22 Disc05: Q2

YouTube link (https://youtu.be/watch?v=BZl73ko7ucg&list=PLx38hZJ5RLZd8OMR9svrGe5nT0-Ycf9v4)

Mutability

Some objects in Python, such as lists and dictionaries, are **mutable**, meaning that their contents or state can be changed. Other objects, such as numeric types, tuples, and strings, are **immutable**, meaning they cannot be changed once they are created.

Let's imagine you order a mushroom and cheese pizza from La Val's, and they represent your order as a list:

```
>>> pizza = ['cheese', 'mushrooms']
```

With list mutation, they can update your order by mutate `pizza` directly rather than having to create a new list:

```
>>> pizza.append('onions')
>>> pizza
['cheese', 'mushrooms', 'onions']
```

Aside from `append`, there are various other list mutation methods:

- `append(el)`: Add `el` to the end of the list. Return `None`.
- `extend(lst)`: Extend the list by concatenating it with `lst`. Return `None`.
- `insert(i, el)`: Insert `el` at index `i`. This does not replace any existing elements, but only adds the new element `el`. Return `None`.
- `remove(el)`: Remove the first occurrence of `el` in list. Errors if `el` is not in the list. Return `None` otherwise.
- `pop(i)`: Remove and return the element at index `i`.

We can also use list indexing with an assignment statement to change an existing element in a list. For example:

```
>>> pizza[1] = 'tomatoes'
>>> pizza
['cheese', 'tomatoes', 'onions']
```

Q1: WWPD: Mutability

What would Python display? In addition to giving the output, draw the box and pointer diagrams for each list to the right.

```
>>> s1 = [1, 2, 3]
>>> s2 = s1
>>> s1 is s2
```

```

```

```
True
```

```
>>> s2.extend([5, 6])
>>> s1[4]
```

```

```

```
6
```

```
>>> s1.append([-1, 0, 1])
>>> s2[5]
```

```

```

```
[-1, 0, 1]
```

```
>>> s3 = s2[:]
>>> s3.insert(3, s2.pop(3))
>>> len(s1)
```

```

```

```
5
```

```
>>> s1[4] is s3[6]
```

```

```

```
True
```

```
>>> s3[s2[4][1]]
```

```
1
```

```
>>> s1[:3] is s2[:3]
```

```
False
```

```
>>> s1[:3] == s2[:3]
```

```
True
```

```
>>> s1[4].append(2)
>>> s3[6][3]
```

```
2
```

Q2: Add This Many

Write a function that takes in a value x , a value el , and a list s , and adds el to the end of s the same number of times that x occurs in s . **Make sure to modify the original list using list mutation techniques.**

Your Answer

```
1    def add_this_many(x, el, s):
2        """ Adds el to the end of s the number of times x occurs in s.
3        >>> s = [1, 2, 4, 2, 1]
4        >>> add_this_many(1, 5, s)
5        >>> s
6        [1, 2, 4, 2, 1, 5, 5]
7        >>> add_this_many(2, 2, s)
8        >>> s
9        [1, 2, 4, 2, 1, 5, 5, 2, 2]
10       """
11       "*** YOUR CODE HERE ***"
12
13
```

Solution

```python
def add_this_many(x, el, s):
    """ Adds el to the end of s the number of times x occurs in s.
    >>> s = [1, 2, 4, 2, 1]
    >>> add_this_many(1, 5, s)
    >>> s
    [1, 2, 4, 2, 1, 5, 5]
    >>> add_this_many(2, 2, s)
    >>> s
    [1, 2, 4, 2, 1, 5, 5, 2, 2]
    """
    count = 0
    for element in s:
        if element == x:
            count += 1
    while count > 0:
        s.append(el)
        count -= 1
```

Two alternate solutions involve iterating over the list indices and iterating over a copy of the list:

```python
def add_this_many_alt1(x, el, s):
    for i in range(len(s)):
        if s[i] == x:
            s.append(el)
```

```python
def add_this_many_alt2(x, el, s):
    for element in list(s):
        if element == x:
            s.append(el)
```

OOP

Object-oriented programming (OOP) is a programming paradigm that allows us to treat data as objects, like we do in real life.

For example, consider the **class** `Student`. Each of you as individuals is an **instance** of this class.

Details that all CS 61A students have, such as `name`, are called **instance variables**. Every student has these variables, but their values differ from student to student. A variable that is shared among all instances of `Student` is known as a **class variable**. For example, the `extension_days` attribute is a class variable as it is a property of all students.

All students are able to do homework, attend lecture, and go to office hours. When functions belong to a specific object, they are called **methods**. In this case, these actions would be methods of `Student` objects.

Here is a recap of what we discussed above:

- **class**: a template for creating objects
- **instance**: a single object created from a class
- **instance variable**: a data attribute of an object, specific to an instance
- **class variable**: a data attribute of an object, shared by all instances of a class
- **method**: a bound function that may be called on all instances of a class

Instance variables, class variables, and methods are all considered **attributes** of an object.

Q3: WWPD: Student OOP

Below we have defined the classes `Professor` and `Student`, implementing some of what was described above. Remember that Python passes the `self` argument implicitly to methods when calling the method directly on an object.

```python
class Student:

    extension_days = 3 # this is a class variable

    def __init__(self, name, staff):
        self.name = name # this is an instance variable
        self.understanding = 0
        staff.add_student(self)
        print("Added", self.name)

    def visit_office_hours(self, staff):
        staff.assist(self)
        print("Thanks, " + staff.name)

class Professor:

    def __init__(self, name):
        self.name = name
        self.students = {}

    def add_student(self, student):
        self.students[student.name] = student

    def assist(self, student):
        student.understanding += 1

    def grant_more_extension_days(self, student, days):
        student.extension_days = days
```

What will the following lines output?

```python
>>> callahan = Professor("Callahan")
>>> elle = Student("Elle", callahan)
```

```
Added Elle
```

```python
>>> elle.visit_office_hours(callahan)
```

```
Thanks, Callahan
```

```
>>> elle.visit_office_hours(Professor("Paulette"))
```

```
Thanks, Paulette
```

```
>>> elle.understanding
```

```
2
```

```
>>> [name for name in callahan.students]
```

```
['Elle']
```

```
>>> x = Student("Vivian", Professor("Stromwell")).name
```

```
Added Vivian
```

```
>>> x
```

```
'Vivian'
```

```
>>> [name for name in callahan.students]
```

```
['Elle']
```

```
>>> elle.extension_days
```

3

```
>>> callahan.grant_more_extension_days(elle, 7)
>>> elle.extension_days
```

7

```
>>> Student.extension_days
```

3

Q4: Keyboard

We'd like to create a `Keyboard` class that takes in an arbitrary number of `Button`s and stores these `Button`s in a dictionary. The keys in the dictionary will be `int`s that represent the position on the `Keyboard`, and the values will be the respective `Button`. Fill out the methods in the `Keyboard` class according to each description, using the doctests as a reference for the behavior of a `Keyboard`.

> **Hint:** You can iterate over *args as if it were a list.

Your Answer

```
1   class Button:
2       def __init__(self, pos, key):
3           self.pos = pos
4           self.key = key
5           self.times_pressed = 0
6
7   class Keyboard:
8       """A Keyboard takes in an arbitrary amount of buttons, and has a
9       dictionary of positions as keys, and values as Buttons.
10      >>> b1 = Button(0, "H")
11      >>> b2 = Button(1, "I")
12      >>> k = Keyboard(b1, b2)
13      >>> k.buttons[0].key
14      'H'
15      >>> k.press(1)
16      'I'
17      >>> k.press(2) # No button at this position
18      ''
19      >>> k.typing([0, 1])
20      'HI'
21      >>> k.typing([1, 0])
22      'IH'
23      >>> b1.times_pressed
24      2
25      >>> b2.times_pressed
26      3
27      """
```

```
28        def __init__(self, *args):
29            _____
30            for _____ in _____:
31                _____
32
33        def press(self, info):
34            """Takes in a position of the button pressed, and
35            returns that button's output."""
36            if _____:
37                _____
38                _____
39                _____
40            _____
41
42        def typing(self, typing_input):
43            """Takes in a list of positions of buttons pressed, and
44            returns the total output."""
45            _____
46            for _____ in _____:
47                _____
48            _____
49
50
```

Solution

```python
class Button:
    def __init__(self, pos, key):
        self.pos = pos
        self.key = key
        self.times_pressed = 0


class Keyboard:
    """A Keyboard takes in an arbitrary amount of buttons, and has a
    dictionary of positions as keys, and values as Buttons.
    >>> b1 = Button(0, "H")
    >>> b2 = Button(1, "I")
    >>> k = Keyboard(b1, b2)
    >>> k.buttons[0].key
    'H'
    >>> k.press(1)
    'I'
    >>> k.press(2) # No button at this position
    ''
    >>> k.typing([0, 1])
    'HI'
    >>> k.typing([1, 0])
    'IH'
    >>> b1.times_pressed
    2
    >>> b2.times_pressed
    3
    """
    def __init__(self, *args):
        self.buttons = {}
        for button in args:
            self.buttons[button.pos] = button

    def press(self, info):
        """Takes in a position of the button pressed, and
        returns that button's output."""
        if info in self.buttons.keys():
            b = self.buttons[info]
            b.times_pressed += 1
            return b.key
        return ''

    def typing(self, typing_input):
        """Takes in a list of positions of buttons pressed, and
        returns the total output."""
        accumulate = ''
        for pos in typing_input:
            accumulate+=self.press(pos)
        return accumulate
```